# 2

# 開發簡單的 Servlet & JSP

學習目標：

- 了解如何撰寫 Servlet 類別
- 了解如何設定部署描述檔
- 了解 Web 應用程式的檔案組織與部署
- 認識 Model 2 架構

# 2.1 從 Servlet 到 Web 容器

要撰寫一個 Servlet 類別很簡單，只要繼承 javax.servlet.http.HttpServlet 類別，並重新定義（override）doGet()、doPost() 等對應 HTTP 請求的方法。然而到這邊爲止，還稱不上一個可以服務的 Servlet，因爲容器還沒有載入這個 Servlet 類別。

你必須設定好 web.xml **部署描述檔（Deployment Descriptor）**，這樣容器才知道要載入以及如何處理你的 Servlet，之後爲你管理 Servlet 實例進行服務。Web 應用程式在檔案組織上也有特定結構，容器會至特定的位置尋找類別或設定檔。

Servlet 的撰寫、web.xml 的設定、Web 應用程式的檔案組織與部署，是本節即將說明的內容。

## 2.1.1 從 Servlet 撰寫認識 HttpServlet

直接來看一個 Servlet 類別如何撰寫：

FirstServlet　　HelloServlet.java

```
package cc.openhome;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {    ←❶ 繼承 HttpServlet
    @Override
    protected void doGet(HttpServletRequest request,    ←❷ 重新定義 doGet()
                         HttpServletResponse response)
                       throws ServletException, IOException {    ❸ 設定回應
        response.setContentType("text/html;charset=UTF-8");    ←┘ 內容類型
        PrintWriter out = response.getWriter();    ←❹ 取得回應輸出物件
        String name = request.getParameter("name");    ←
                                                          ❺ 取得請求參數
```

```
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1> Hello! " + name + " !</h1>");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

❻ 跟使用者說 Hello!

範例中繼承了 HttpServlet❶，並重新定義了 doGet() 方法❷，當瀏覽器時 GET 方法發送請求時，會呼叫此方法。

在 doGet() 方法上可以看到 HttpServletRequest 與 HttpServletResponse 兩個參數，容器接收到客戶端的 HTTP 請求後，會收集 HTTP 請求中的資訊，並分別建立代表請求與回應的 Java 物件，而後在呼叫 doGet() 時將這兩個物件當作參數傳入。你可以從 HttpServletRequest 物件中取得有關 HTTP 請求相關資訊，在範例中是透過 HttpServletRequest 的 getParameter() 並指定請求參數名稱，來取得使用者所發送的請求參數值❺。

注意 範例中的 @Override 是 JDK5 之後所提供的 Annotation，作用是協助檢查是否正確地重新定義父類別中所繼承下來的方法，就撰寫 Servlet 而言，沒有 @Override 並沒有影響。

HttpServletResponse 物件代表對客戶端的回應，可以藉由其 setContentType() 設定正確的內容類型❸，範例中是告知瀏覽器，傳回的回應要以 text/html 解析，而採用的字元編碼是 UTF8。接著再使用 getWriter() 方法取得代表回應輸出的 PrintWriter 物件❹，藉由 PrintWriter 的 println() 方法來對瀏覽器輸出回應的文字資訊，在範例中是輸出 HTML 以及根據使用者名稱說聲 Hello!❻。

提示 學習 Servlet／JSP 時有一些專案設定與部署的細節，建議你選擇一個**整合開發環境**（Integrated Development Environment, IDE）來撰寫程式。為了初學者學習上的方便，本書將提供 Eclipse 與 NetBeans IDE

的專案，並使用 Tomcat 來作為 Web 容器。若你不熟悉 Eclipse 或 NetBeans IDE，附錄中分別包含了 Eclipse 與 NetBeans IDE 的使用簡介可供參考。

藉由這個範例，再進一步來思考一個問題，爲什麼要在繼承 `HttpServlet` 之後重新定義 `doGet()`，又爲什麼 HTTP 請求爲 GET 時會自動呼叫 `doGet()`。首先來討論範例中所看到的應用程式介面（Application Interface, API）類別圖：

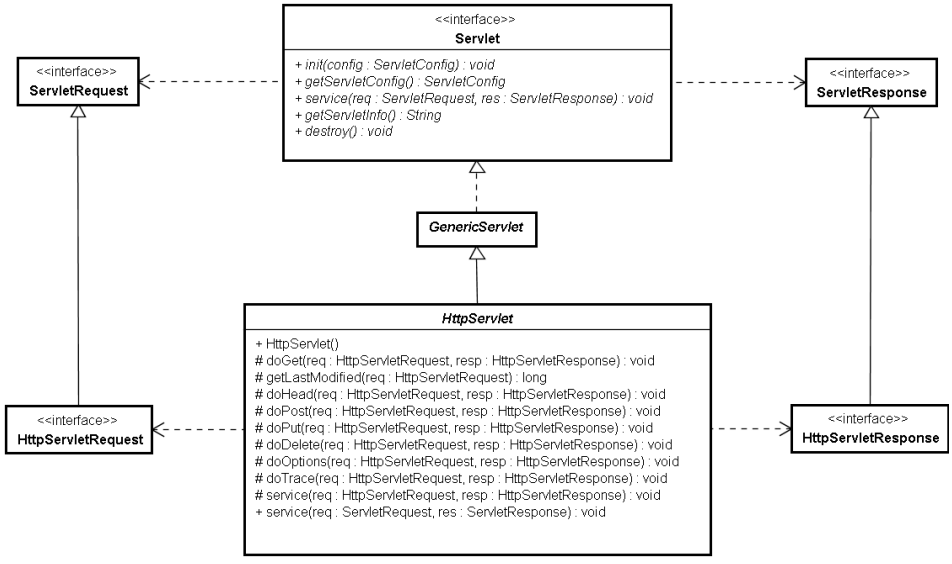



圖 2.1 HttpServlet 相關 API 類別圖

首先看到 `Servlet` 介面，它定義了 Servlet 所應當有的基本行爲，例如與 Servlet 生命週期相關的 `init()`、`destroy()` 方法、提供服務時所要呼叫的 **`service()`** 方法等。

實作 `Servlet` 介面的類別是 `GenericServlet` 類別，它還實作了 `ServletConfig` 介面，將容器呼叫 `init()` 方法時所傳入的 `ServletConfig` 實例封裝起來，而 `service()` 方法直接標示爲 `abstract` 而沒有任何的實作。在本章中將暫且忽略對 `GenericServlet` 的討論，只需先知道有它的存在（第 4 章會加以討論）。

在這邊只要先注意到一件事，GenericServlet 並沒有規範任何有關 HTTP 的相關方法，而是由繼承它的 HttpServlet 來定義。在最初定義 Servlet 時，並不限定它只能用於 HTTP，所以並沒有將 HTTP 相關服務流程定義在 GenericServlet 之中，而是定義在 HttpServlet 的 service() 方法中。

提示 你可以注意到套件（package）的設計，與 Servlet 定義相關的類別或介面都位於 javax.servlet 套件之中，像是 Servlet、GenericServlet、ServletRequest、ServletResponse 等。而與 HTTP 定義相關的類別或介面都位於 javax.servlet.http 套件之中，像是 HttpServlet、HttpServletRequest、HttpServletResponse 等。

以 Tomcat 實作爲例，HttpServlet 的 service() 方法中的流程大致如下：

```
protected void service(HttpServletRequest req,
                       HttpServletResponse resp)
    throws ServletException, IOException {
    String method = req.getMethod(); // 取得請求的方法
    if (method.equals(METHOD_GET)) { // HTTP GET
        // 略...
        doGet(req, resp);
        // 略 ...
    } else if (method.equals(METHOD_HEAD)) { // HTTP HEAD
        // 略 ...
        doHead(req, resp);
    } else if (method.equals(METHOD_POST)) { // HTTP POST
        // 略 ...
        doPost(req, resp);
    } else if (method.equals(METHOD_PUT)) { // HTTP PUT
        // 略 ...
}
```

當請求來到時，容器會呼叫 Servlet 的 service() 方法，而可以看到，HttpServlet 的 service() 中所定義的，基本上就是判斷 HTTP 請求的方式，再分別呼叫 doGet()、doPost() 等方法，所以若想針對 GET、POST 等方法進行處理，才會只需要在繼承 HttpServlet 之後，重新定義相對應的 doGet()、doPost() 方法。

注意 這其實是使用了設計模式（Design Pattern）中的 **Template Method 模式**。所以不建議也不應該在繼承了 `HttpServlet` 之後，重新定義 `service()` 方法，這會覆蓋掉 `HttpServlet` 中所定義的 HTTP 預設處理流程。

## 2.1.2 設定部署描述檔

撰寫好 Servlet 之後，接下來要告訴 Web 容器有關於這個 Servlet 的一些資訊。Web 容器會讀取一個檔名爲 web.xml 的部署描述檔，你的 Servlet 相關資訊就是定義在這個檔案之中，例如在 web.xml 中定義先前的 `HelloServlet`。

FirstServlet web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ➥xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  ➥http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>cc.openhome.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello.do</url-pattern>
    </servlet-mapping>

</web-app>
```

**<web-app>** 開頭標籤中的東西，主要是一些與 Servlet 版本、XML 有關的資訊，你不用記憶這些東西，使用 IDE 時基本上也會自動產生這些開頭標籤的內容。最主要的是注意到 **<servlet>** 與 **<servlet-mapping>** 標籤及其中相關標籤的設定。

**<servlet>** 標籤中的 **<servlet-class>** 定義告訴容器應該載入哪個類別檔案，在這個例子中是載入 `cc.openhome.HelloServlet`，注意要使用類

別的**完全符合名稱**（Full Qualified Name），也就是包含套件名稱與類別名稱的完整名稱。**<servlet-name>** 標籤中的定義會告訴容器，爲該類別建立一個實例，並註冊爲 HelloServlet 這個名稱，這個名稱可以任意命名（在這邊只是剛好與類別名稱相同）。

接下來要告訴容器，客戶端請求哪個 URL 時，要由哪個 Servlet 實例來進行處理，這是定義在 **<servlet-mapping>** 標籤中。**<url-pattern>** 中設定 /hello.do，而 <servlet-name> 定義爲 HelloServlet，這表示當客戶端請求應用程式的 /hello.do 這個 URL 時，Web 容器會對應設定並交由註冊爲 HelloServlet 名稱的 Servlet 實例來處理請求。圖 2.2 是從請求至交由 Servlet 實例處理的對應流程：

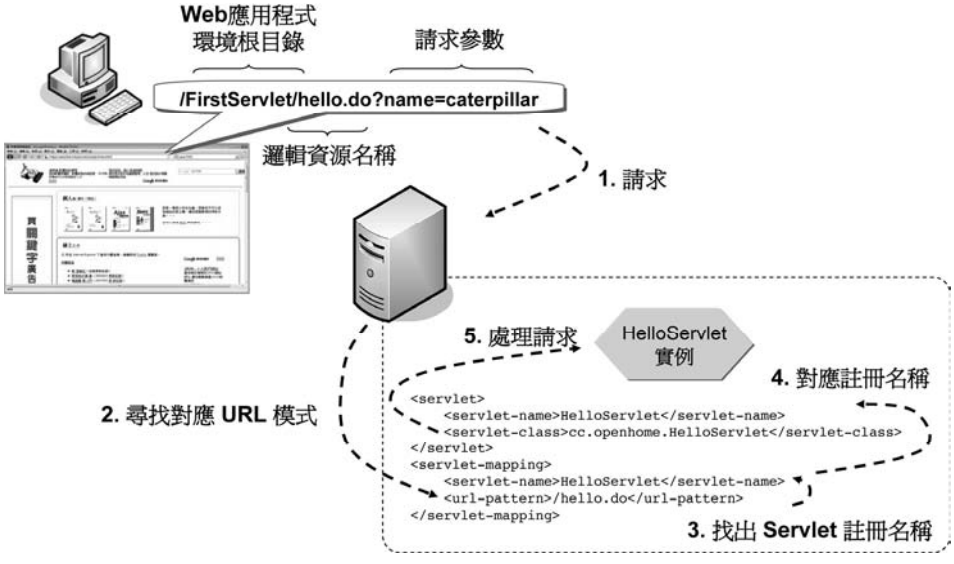



圖 2.2　Servlet 的請求對應

在圖 2.2 中，Web 應用程式**環境根目錄**（Context Root）是可以自行設定，不過設定方式會因所使用的 Web 應用程式伺服器而有所不同。例如 Tomcat 預設會使用應用程式目錄作爲環境根目錄。簡單地說，如果你的應用程式環境根目錄是 FirstServlet，則必須以下列 URL 進行請求（其中 8080 爲 Tomcat 所預設的連接埠號）：

http://localhost:8080/FirstServlet/hello.do?name=caterpillar

而從圖 2.2 中也可以知道，請求時的 URL 是個**邏輯名稱（Logical Name）**，因爲 /hello.do 並不是指伺服器上真的有個實體檔案叫 hello.do，而會再由 Web 容器對應至實際處理請求的檔案或程式**實體名稱（Physical Name）**。如果你願意，也可以在 <url-pattern> 上用個像 hello.jsp 之類的名稱來僞裝你的資源。

因而到目前爲止，你可以知道，一個 Servlet 在 web.xml 中會有三個名稱設定：<url-pattern> 設定的邏輯名稱、<servlet-name> 註冊的 Servlet 名稱、以及 <servlet-class> 設定的實體類別名稱。

## 2.1.3 Web 應用程式檔案組織

撰寫完 Servlet 與 web.xml 之後，接著就是編譯 Servlet，並將檔案組織爲 Web 應用程式的目錄及檔案結構。

要自行編譯 Servlet（IDE 會自動編譯），你的類別路徑（Classpath）中必須包括 Servlet API 的相關類別，如果使用的是 Tomcat，則這些類別通常是封裝在 Tomcat 目錄的 lib 目錄中的 servlet-api.jar。假設你的 HelloServlet.java 位於 src 目錄下，並放置於對應套件的目錄之中，則你可以如下進行編譯：

```
% cd YourWorkspace/FirstServlet
% javac -classpath Yourlibrary/YourTomcat/lib/servlet-api.jar -d ./classes
src/cc/openhome/HelloServlet.java
```

注意底線部份必須修改爲你實際的目錄位置，編譯出的 .class 檔案會出現於 classes 目錄中，並有對應的套件階層（因爲使用 javac 時下了 -d 引數）。接下來編譯好的 .class 檔案與 web.xml，必須如下組織：

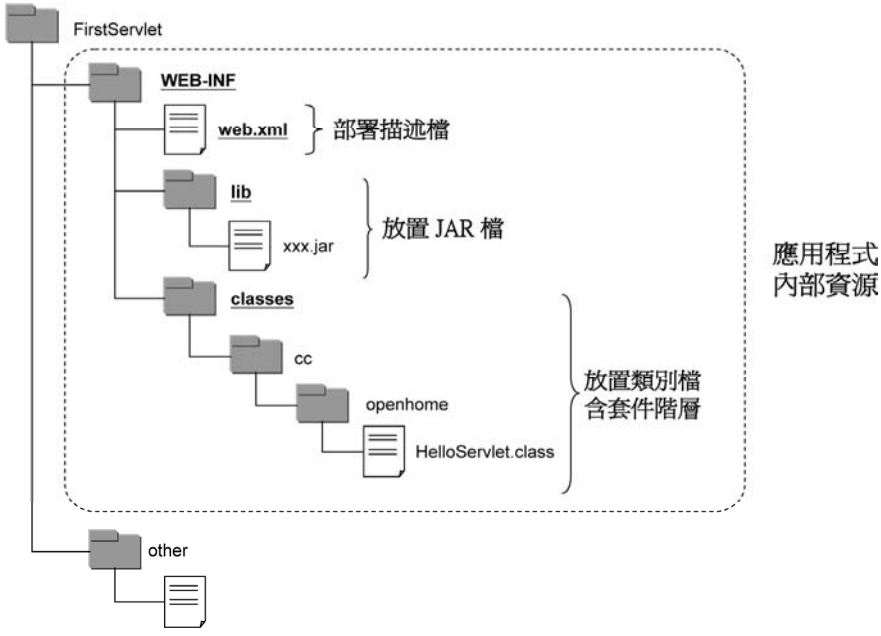



圖 2.3 Web 應用程式檔案組織

圖 2.3 有幾個重要的目錄與檔案位置必須說明：

- WEB-INF

  這個目錄名稱是固定的，而且一定是位於應用程式根目錄下，放置在 WEB-INF 中的檔案或目錄，對外界來說是封閉的，也就是客戶端無法使用 HTTP 的任何方式直接存取到 WEB-INF 中的檔案或目錄。若有這類需要，則必須透過 Servlet／JSP 的請求轉發（Forward）。你不想讓外界直接存取的資源，可以放置在這個目錄下。

- web.xml

  這是 Web 應用程式部署描述檔，一定是放在 WEB-INF 根目錄下，名稱一定是 web.xml。

- lib

  放置 JAR（Java Archive）檔案的目錄，一定是放在 WEB-INF 根目錄下，名稱一定是 lib。

- classes

  放置編譯過後 .class 檔案的目錄，一定是放在 WEB-INF 目錄下，名稱一定是 classes。編譯過後的類別檔案，必須有與套件名稱相符的目錄結構。

如果你使用 Tomcat 作爲 Web 容器，則可以將圖 2.3 的 FirstServlet 整個目錄複製至 Tomcat 目錄下 webapps 目錄，然後至 Tomcat 的 bin 目錄下，執行 `catalina run` 指令來啓動 Tomcat，接著就可以用以下的 URL 請求應用程式：

http://localhost:8080/FirstServlet/hello.do?name=caterpillar


圖 2.4 第一個 Servlet 應用程式

## 2.1.4 WAR 的建立與部署

實際上在部署 Web 應用程式時，會將 Web 應用程式封裝爲一個 WAR（Web Archive）檔案，也就是一個副檔名爲 *.war 的檔案。WAR 檔案可使用 JDK 所附的 jar 工具程式來建立。例如，當你如圖 2.3 的方式組織好 Web 應用程式檔案之後，可進入 FirstServlet 目錄，然後執行以下指令：

```
jar cvf ../FirstServlet.war *
```

這會在你的 FirstServlet 目錄外建立一個 FirstServlet.war 檔案，WAR 檔案是使用 zip 壓縮格式封裝，可以使用解壓縮軟體來檢視其中的內容。如果使用 Tomcat，則可以將所建立的 WAR 檔案複製至 webapps 目錄下，重新啓動 Tomcat，容器若發現 webapps 目錄中有 WAR 檔案，會將之解壓縮，並載入 Web 應用程式。

不同的應用程式伺服器，會提供不同的指令或介面讓你部署 WAR 檔案。如果以 Tomcat 爲例，其附帶一個簡單的管理介面讓你部署 Web 應用程式：

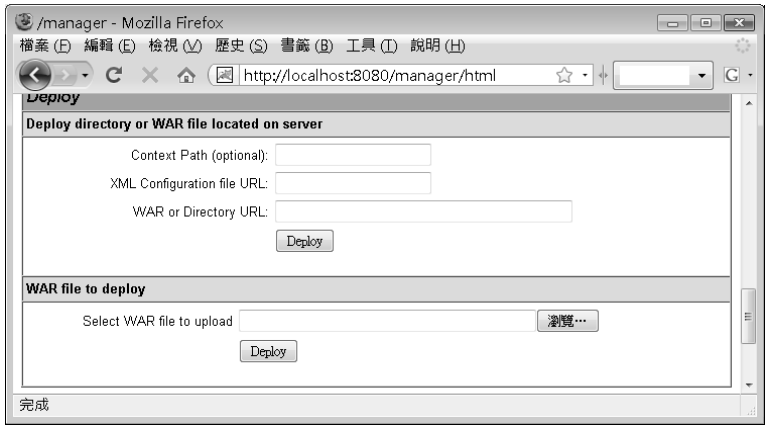



圖 2.5　Tomcat 所附帶的 Web 應用程式部署管理介面

提示　使用 Tomcat 的話，可以連接至 http://localhost:8080，點選「Tomcat Manager」登入管理介面，但在這之前，記得設定管理者名稱與密碼再啟動 Tomcat，這要在 Tomcat 目錄下 conf 目錄的 tomcat-users.xml 設定，例如若要設定使用者名稱為 tomcat、密碼為 tomcat：

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager"/>
    <user username="tomcat" password="tomcat" roles="manager"/>
</tomcat-users>
```

## 2.2　在 HelloServlet 之後

即使是個簡單的 HelloServlet，在載入 Web 容器之後，Web 容器也爲它作了許多的處理細節，至少你應該懷疑一下 `doGet()` 上的兩個參數是怎麼來的。

在 Servlet 中用字串方式撰寫 HTML，然後再透過從 `HttpServletResponse` 取得的 `PrintWriter` 物件進行輸出，這麼作絕對不

會是什麼好事。在開發 Servlet／JSP 撰寫的 Web 應用程式時，你必須知道 Model 2 架構是什麼，並作好邏輯與畫面的職責分配。

本節將再深入一點看看，Web 容器爲你作了些什麼，還有解釋何謂 Model 2 架構，並看看一個簡單的 Model 2 架構程式。

## 2.2.1 Web 容器作了什麼？

在上一節中，已經看過 Web 容器爲你作的事情就是，建立 Servlet 實例，並完成 Servlet 名稱註冊及 URL 模式的對應。在請求來到時，Web 容器會轉發給正確的 Servlet 來處理請求。

當瀏覽器請求 HTTP 伺服器時，是使用 HTTP 來傳送請求與相關資訊（標頭、請求參數、Cookie 等）。HTTP 是基於 TCP/IP 之上的協定，資訊基本上都是透過文字訊息來傳送，然而 Servlet 本質上是個 Java 物件，運行於 Web 容器（一個 Java 寫的應用程式）之中。有關於 HTTP 請求的相關訊息，如何變成相對應的 Java 物件呢？

當請求來到 HTTP 伺服器，而 HTTP 伺服器轉交請求給容器時，容器會建立一個代表當次請求的 `HttpServletRequest` 物件，並將請求相關資訊設定給該物件。同時，容器會建立一個 `HttpServletResponse` 物件，作爲稍後要對客戶端進行回應的 Java 物件。

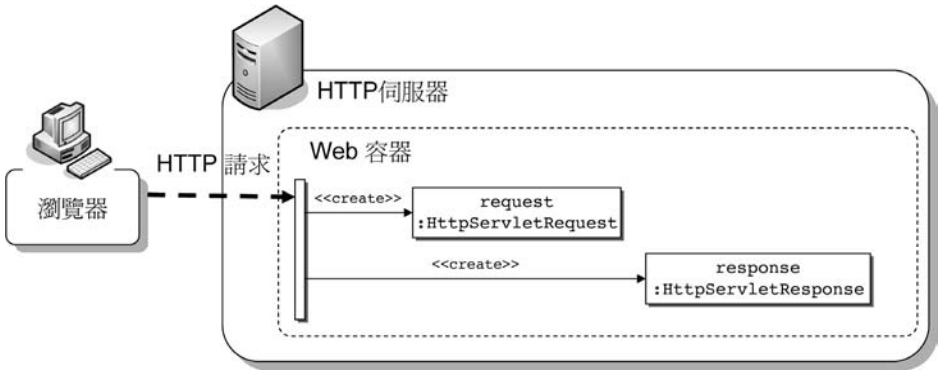



圖 2.6 容器爲你收集相關資訊，並建立代表請求與回應的 Java 物件

接著，容器會根據 web.xml 的設定，找出處理該請求的 Servlet，呼叫它的 service() 方法，將所建立的 HttpServletRequest 物件、HttpServletResponse 物件傳入作爲參數，service() 方法中會根據 HTTP 請求的方式，呼叫對應的 doXXX() 方法，例如若爲 GET，則呼叫 doGet() 方法。

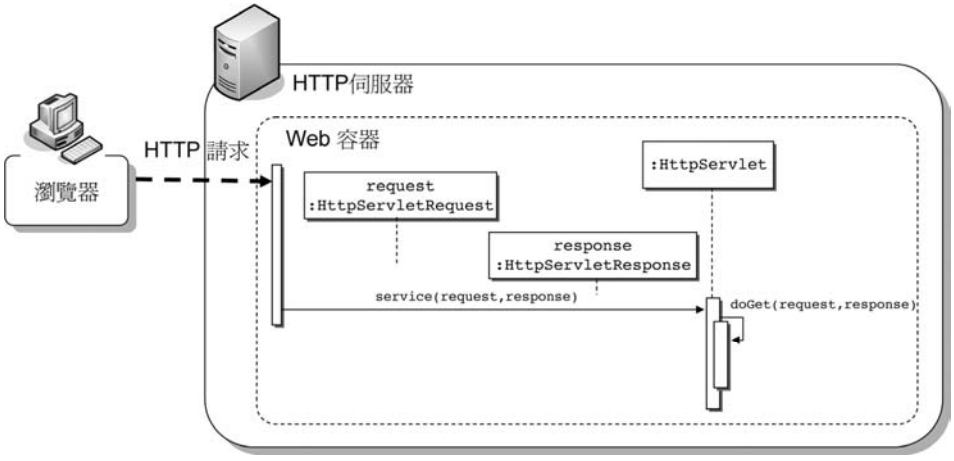



圖 2.7　容器呼叫 Servlet 的 **service()** 方法

接著在 doGet() 方法中，你可以使用 HttpServletRequest 物件、HttpServletResponse 物件，例如使用 getParameter() 取得請求參數，使用 getWriter() 取得輸出用的 PrintWriter 物件，並進行各項回應處理。你對 PrintWriter 所作的輸出操作，最後會由容器爲你轉換爲 HTTP 回應，然後再由 HTTP 伺服器對瀏覽器進行回應。之後容器將 HttpServletRequest 物件、HttpServletResponse 物件銷毀回收，該次請求回應結束。

沒有了 Web 容器，請求資訊的收集、HttpServletRequest 物件、HttpServletResponse 物件等的建立、輸出 HTTP 回應之轉換、HttpServletRequest 物件、HttpServletResponse 物件等的銷毀、回收等，都必須自己動手完成（你可以想像自行用 Java SE 撰寫 HTTP 伺服器，並完成這些功能有多麻煩）。有了容器提供這些服務（當然還有更多服務，之後章節還會陸續提到），你就可以專心在 Java 物件之間的互動來解決問題。

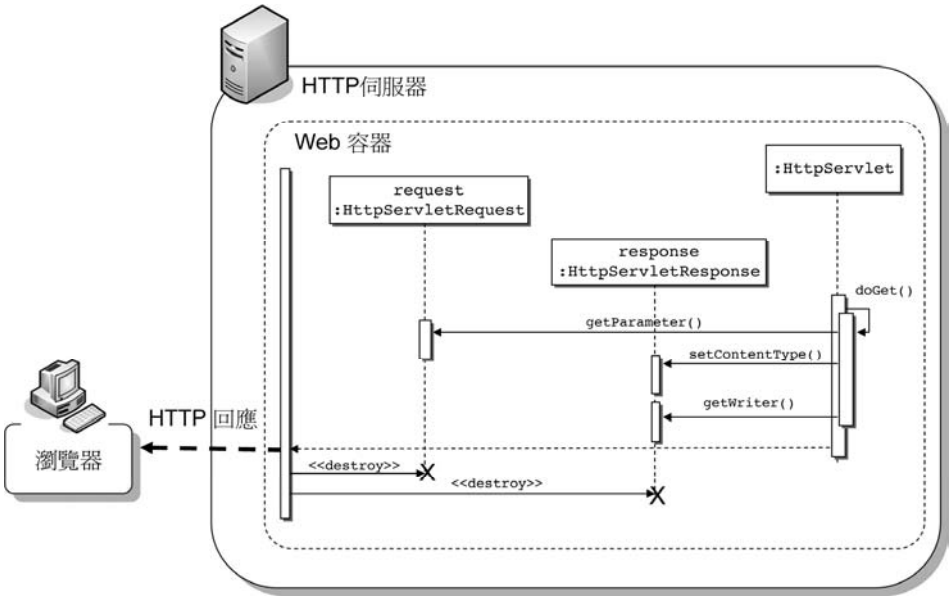



圖 2.8 容器爲你轉換 HTTP 回應，並銷毀、回收當次請求回應等相關物件

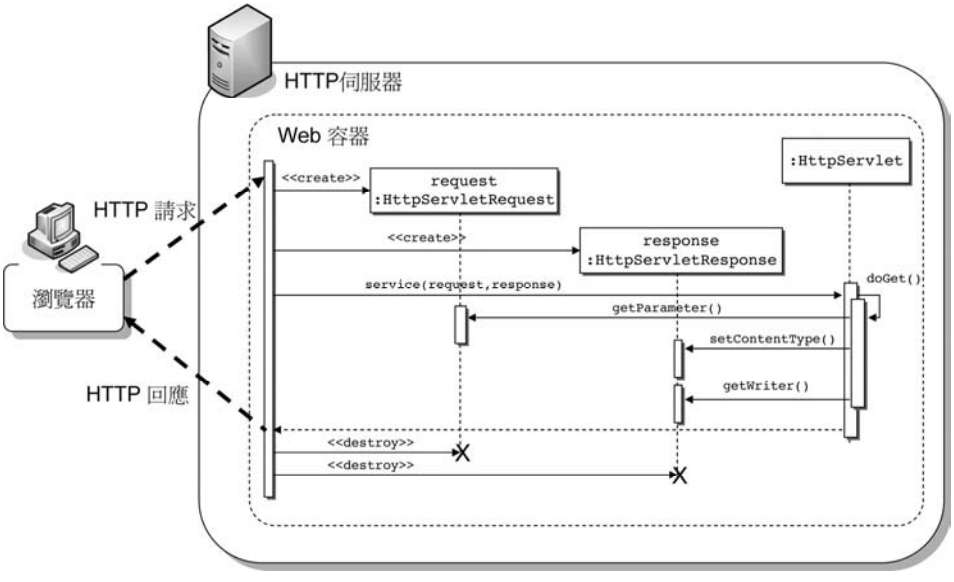



圖 2.9 從請求到回應，容器內所提供的服務流程示意

## 2.2.2 簡介 MVC 與 Model 2

無論如何！在 Servlet 程式中夾雜 HTML 的畫面輸出絕對不是什麼好主意，而在之後介紹到 JSP 時，你會知道 JSP 中也可以撰寫 Java 程式碼，但在 JSP 網頁中的 HTML 間夾雜 Java 程式邏輯，也是極度不建議的作法。Java 程式語法邏輯與呈現畫面的 HTML 等攪和在一起，一來撰寫不易、二來日後維護不易、三來對團隊間的分工合作也是一大困擾。

在談及 Web 應用程式架構上的設計時，總會談到 MVC 或 Model 2 這兩個名詞。MVC 是 Model、View、Controller 的縮寫，分別代表應用程式中三種職責各不相同的物件。

- Model

  中文稱之爲「**模型**」，就是**封裝了應用程式功能或狀態的物件**。設計上建議模型物件必須與畫面所採取的解決方案無關，與底層所使用的存取機制無關，要是一個中性的物件。因爲模型是應用程式的商務（Business）邏輯元件，希望可以具備高可攜性，也就是必要時，希望在最少的修改下就能移植至另一個平台。模型不負責繪製畫面，但在狀態改變時會通知感興趣的「視圖」物件。

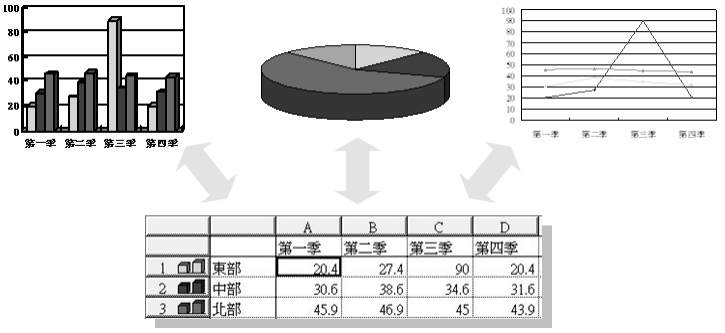
- View

  中文稱之爲「**畫面**」或「**視圖**」，也就是**負責使用者所觀看及操作的介面**。像 Swing 視窗程式或是瀏覽器中的 HTML 頁面，都屬於「視圖」元件。視圖會對感興趣的模型進行註冊，如此在模型狀態改變時可收到通知，而後查詢模型的最新狀態並更新畫面。

- Controller

  中文稱之爲「**控制器**」，職責在於**收集使用者請求的相關資訊，並轉發給對應的模型物件**。控制器可能操作某個商務模型，或者是修改某個資料模型。

以下舉 MVC 的架構的一個應用範例。想像你有一個表格，該表格的資料可以使用圓餅圖來呈現，也可以用長條圖來呈現，或是用折線圖來呈現。




| | | A | B | C | D |
|---|---|---|---|---|---|
| | | 第一季 | 第二季 | 第三季 | 第四季 |
| 1 | 東部 | 20.4 | 27.4 | 90 | 20.4 |
| 2 | 中部 | 30.6 | 38.6 | 34.6 | 31.6 |
| 3 | 北部 | 45.9 | 46.9 | 45 | 43.9 |

圖 2.10 同一個表格的不同視圖呈現

你希望某個使用者可透過所看到的畫面，進行操作來調整資料，因而造成表格資料的變更時（例如調整長條圖），其它視圖也可以收到通知而自動更新顯示狀態（例如圓餅圖或折線圖），這時就可以試著套用 MVC 架構。表格資料可當作是模型物件，而長條圖等畫面則是視圖物件，而程式中所撰寫回應使用者動作的物件，就是屬於控制器物件。

將以上的範例套用 MVC 架構，可用下圖來表示：

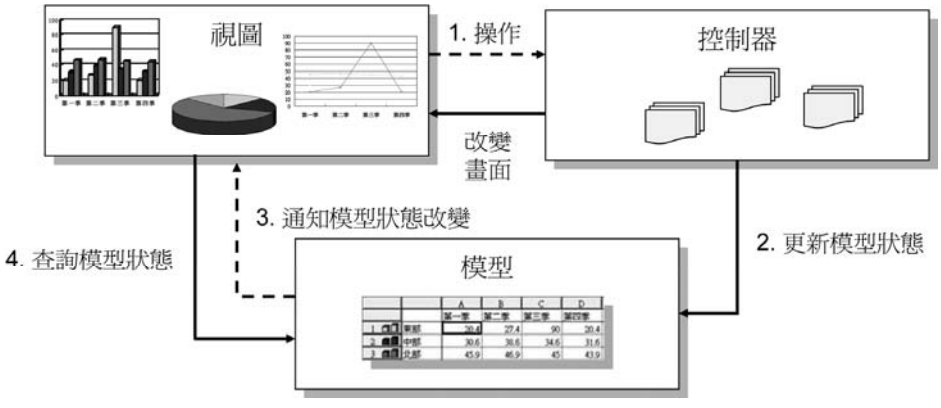



圖 2.11 MVC 互動示意圖

假設一開始，長條圖、圓餅圖、折線圖都對模型物件進行註冊。接下來若在長條圖上進行操作（如圖標號 1 所示），則控制器會看看該操作應當更新哪個模型物件（如標號 2 所示）。模型被控制器更新之後，會逐一通知先前已註冊的視圖物件（如圖標號 3 所示）。被通知的視圖會回過頭來查詢模型的最新狀態（如圖標號 4 所示）並更新自身畫面。

（視圖上的操作也可能是切換畫面顯示，例如切換立體長條圖爲平面長條圖，這可由控制器直接對視圖進行切換操作，如圖 2.11 中無標號的「改變畫面」箭頭。）

所以 MVC 架構的主要目的，就是藉由控制器的請求轉發，切斷模型與視圖之間的耦合關係，讓模型中不會有呈現畫面的相關邏輯，而視圖需要資料時是經由查詢模型的方式來取得。

回過頭來思考前一節介紹 Servlet 撰寫時，Java 程式碼中夾雜 HTML 的情況，你應該可以聯想到，是否可以使用 MVC 這樣的架構，重新設計物件的職責，讓 Java 程式碼與 HTML 分離，答案是肯定的。此時，模型就是伺服器上的 Java 物件，視圖可以用 JSP 來呈現，控制器則是個接受使用者請求的 Servlet。

只不過將 MVC 套用至 Web 應用程式時必須作一些修正，因爲 HTTP 在天性上有個關鍵性的不同，HTTP 是基於請求／回應的通訊協定，在模型狀態改變時，你無法從 HTTP 伺服器對瀏覽器發出通知，以要求瀏覽器查詢模型並更新畫面。沒有請求，HTTP 伺服器就不會有回應。簡單地說，你無法在 HTTP 上作到圖 2.11 上標號爲 4 的動作。

所以，若瀏覽器需要取得模型的最新狀態，必須主動發出請求，而不是由模型對瀏覽器進行通知，經由這個修正之後所得到的架構，稱之爲 Model 2 架構。

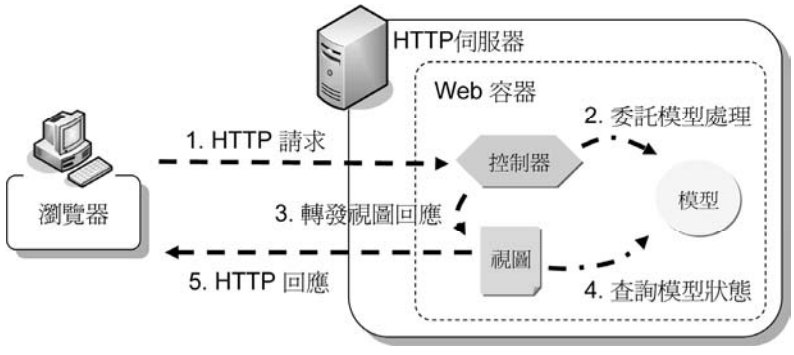



圖 2.12　基於請求／回應進行修正所得到的 Model 2 架構

圖 2.12 以實際的流程說明了 Model 2 三個角色的職責，當瀏覽器對某個 URL 發出 HTTP 請求時（如圖標號 1）：

- 控制器（Controller）

  在 Model 2 架構上，控制器的職責在於**收集請求相關資訊**。例如在圖 2.12 中，就是收集瀏覽器發出的請求參數、標頭、Cookie 等。控制器本身不從事商務邏輯處理，而是委託給模型物件（如圖標號 2），並將收集而到的請求資訊設定給模型物件進行處理。

  當模型處理完畢而執行流程回到控制器時，控制器也不直接作畫面呈現的處理或對客戶端進行回應，而是將流程轉發（Forward）給視圖物件（如圖標號 3）。

  在 Servlet／JSP 應用程式中，通常由 Servlet 技術來實現模型的角色職責。

- 模型（Model）

  模型物件可能是**封裝應用程式狀態或是處理商務邏輯的物件**。它也可能會與後端的資料庫進行互動。模型接收控制器的請求資訊設定，並進行相關商務邏輯處理或狀態變更。

  實現模型角色的物件，通常是個**純粹的 Java 物件（Plain Old Java Object, POJO）**，與呈現畫面的技術無關，與底層儲存技術無關。具體來說，模型物件中不出現與畫面技術或底層儲存技術相關的 API。

- 視圖（View）

  當控制器將流程轉發至視圖時，視圖會**進行畫面的組織**。例如準備 HTML，並從模型物件中查詢必要資訊（如圖標號 4）再進行最後結果輸出（如圖標號 5），瀏覽器收到 HTML 之後，再於瀏覽器中作畫面的呈現。

  在 Servlet／JSP 應用程式中，通常是由 JSP 技術來實現視圖的角色職責。

圖 2.13 是以 Servlet／JSP 技術來呈現 Model 2 架構時，各技術各自適合實現哪個角色：

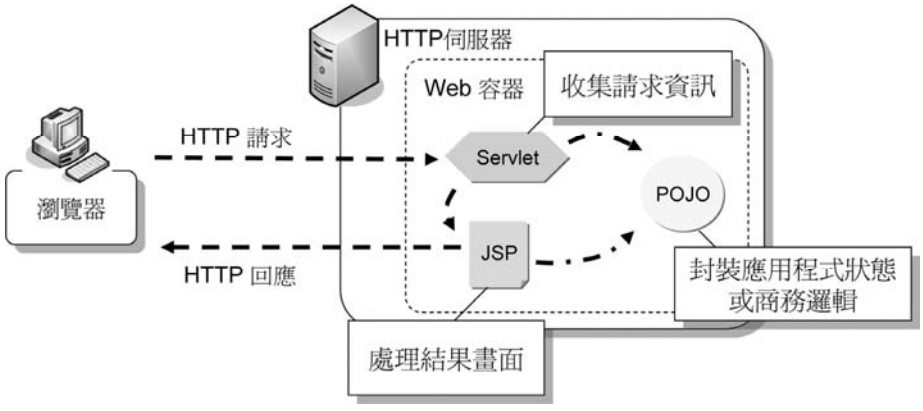



圖 2.13　基於 Servlet／JSP 實現 Model 2 架構

## 2.2.3 簡單的 Model 2 程式

爲了讓你對 Model 2 架構有個具體的印象，接下來將以一個簡單的程式示範 Model 2 架構的實現。這個 Web 應用程式會接受瀏覽器的請求，根據所提供的 user 請求參數顯示不同的歡迎訊息。首先看到控制器，由一個 Servlet 來實現：

SimpleModel2　　HelloServlet.java

```
package cc.openhome;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
    private Hello hello;

    public HelloServlet() {
        hello = new Hello();
    }
```

```
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                     throws ServletException, IOException {
        String name = request.getParameter("user");          ←❶ 收集請求參數
        String message = hello.doHello(name);                ←❷ 委託 Hello 物件處理
        request.setAttribute("message", message);            ←❸ 將結果訊息設定至
        request.getRequestDispatcher("hello.jsp")                請求物件成爲屬性
               .forward(request, response);   ←┐
    }                                          ❹ 轉發給 hello.jsp 進行回應
}
```

即使是這個簡單的程式要實現 Model 2 架構，也需要一些你目前還沒有學習到的 Servlet／JSP 相關知識。如果這個範例中有些程式片段你還不了解，不用擔心，之後的章節將會陸續提到。

就上面這個 Servlet 來說，目前只要注意到，Servlet 會收集請求參數❶並委託一個 `Hello` 物件處理❷，Servlet 中不會有任何 HTML 的出現。`Hello` 物件處理的結果，會設定爲請求物件中的屬性❸，之後呈現畫面的 JSP 可以從請求物件中取得該屬性。接著將請求的回應工作轉發給 hello.jsp 來負責❹。

至於 `Hello` 類別的設計很簡單，利用一個 `HashMap`，針對不同的使用者設定不同的訊息：

SimpleModel2 — Hello.java

```
package cc.openhome;

import java.util.*;

public class Hello {
    private Map<String, String> messages;

    public Hello() {
        messages = new HashMap<String, String>();
        messages.put("caterpillar", "Hello");
        messages.put("Justin", "Welcome");
        messages.put("momor", "Hi");
    }
```

```
    public String doHello(String user) {
        String message = messages.get(user);
        return message + ", " + user + "!";
    }
}
```

這是個再簡單不過的類別。要注意的是，Hello 物件處理完的結果傳回給 Servlet，Hello 類別中不會有任何 HTML 的出現。也沒有任何與 JSP（或後端儲存技術）的 API 出現，是個純粹的 Java 物件。

Servlet 得到 Hello 物件的傳回值之後，將流程轉發給 JSP，由 JSP 呈現畫面：

SimpleModel2　hello.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>${param.user}</title>   ←❶ 利用 Expression Language
    </head>                                取得 user 請求參數
    <body>
        <h1>${message}</h1>   ←❷ 利用 Expression Language
    </body>                        取得請求範圍中設定的屬性值
</html>
```

這個 JSP 網頁中動態的部份，是利用 JSP 2.0 中的 Expression Language 功能，分別取得 user 請求參數❶以及先前 Servlet 中設定在請求物件中的 message 屬性❷。JSP 中沒有任何 Java 程式碼的出現。先來看一下一個執行時的結果畫面：


圖 2.14　根據請求參數呈現不同的結果畫面

之後介紹 JSP 時，你還會看到更多有關 Expression Language 與 JSP 標籤（Tag）的介紹，這些 JSP 中的技術，都是讓網頁編輯人員可以使用自己熟悉的標籤語法，設定 JSP 頁面中必須動態呈現的部份，而不用撰寫 Java 程式碼。

當然！這只是個簡單的示範，主要目的在讓你對 Model 2 的實現有個基本的了解，之後的章節範例將會以 Model 2 架構，逐步實現一個功能更完整的應用程式。

## 2.3 重點複習

要撰寫 Servlet 類別，必須繼承 `HttpServlet` 類別，並重新定義 `doGet()`、`doPost()` 等對應 HTTP 請求的方法。容器會分別建立代表請求、回應的 `HttpServletRequest` 與 `HttpServletResponse`，你可以從前者取得所有關於該次請求的相關資訊，從後者對客戶端進行各種回應。

在 Servlet 的 API 定義中，`Servlet` 是個介面，當中定義了與 Servlet 生命週期相關的 `init()`、`destroy()` 方法，以及提供服務的 `service()` 方法等。`GenericServlet` 實作了 `Servlet` 介面，不過它直接將 `service()` 標示爲 `abstract`，`GenericServlet` 還實作了 `ServletConfig` 介面，將容器初始化 Servlet 呼叫 `init()` 時所傳入的 `ServletConfig` 封裝起來。

真正在 `service()` 方法中定義了 HTTP 請求基本處理流程是 `HttpServlet`，而 `doGet()`、`doPost()` 中傳入的參數是 `HttpServletRequest`、`HttpServletResponse`，而不是通用的 `ServletReqeust`、`ServletResponse`。

要讓 Web 容器知道該載入哪個 Servlet，以及請求時由哪個 Servlet 進行處理，這些必須定義在部署描述檔 web.xml 中。一個 Servlet 至少會有三個名稱，即類別名稱、註冊的 Servlet 名稱與 URL 模式（Pattern）名稱。這三個名稱的對應是使用標籤 `<servlet>`（子標籤 `<servlet-name>`、`<servlet-class>`）與 `<servlet-mapping>`（子標籤 `<servlet-name>`、`<url-pattern>`）來定義。

Web 應用程式有幾個要注意的目錄與結構，WEB-INF 中的資料客戶端無法直接請求取得，而必須透過請求的轉發才有可能存取。web.xml 必須位於 WEB-INF 中。lib 目錄用來放置 Web 應用程式會使用到的 JAR 檔案。classes 目錄用來放置編譯好的 .class 檔案。你可以將整個 Web 應用程式使用到的所有檔案與目錄封裝爲 WAR（Web Archive）檔案，即副檔名爲 .war 的檔案，再利用 Web 應用程式伺服器所提供的工具來進行應用程式的部署。

到這個章節爲止，你知道 Web 容器可以載入 Servlet、管理 Servlet 生命週期、爲 HTTP 請求建立分別代表請求與回應的 `HttpServletRequest` 與 `HttpServletResponse`，而在回應完畢後，再分別銷毀 `HttpServletRequest` 與 `HttpServletResponse` 物件。

MVC 架構所代表的，是將應用程式區分爲模型（Model）、視圖（View）、控制器（Controller）三大角色職責。模型封裝了應用程式功能或狀態的物件。「視圖」負責使用者所觀看及操作的介面。控制器收集使用者請求的相關資訊，並轉發給對應的模型物件。當模型的狀態有所更新時，會通知當初有註冊的視圖物件，視圖物件再對模型物件的進行狀態查詢以及自身畫面的更新。

基於 HTTP 請求／回應的 Web 應用程式，無法達成 MVC 架構中模型通知視圖的功能，因而作了適當修正，所得到的就是 Model 2 架構。在 Servlet／JSP 的實現中，控制器通常用 Servlet 擔任，模型是個純粹 Java 物件，而視圖則由 JSP 來實現。

# 課後練習

## 選擇題

1. 若要針對 HTTP 請求撰寫 Servlet 類別，以下何者是正確的作法？

   (A) 實作 `Servlet` 介面

   (B) 繼承 `GenericServlet`

   (C) 繼承 `HttpServlet`

   (D) 直接定義一個結尾名稱爲 Servlet 的類別

2. 續上題，如何針對 HTTP 的 GET 請求進行處理與回應？

   (A) 重新定義 `service()` 方法

   (B) 重新定義 `doGet()` 方法

   (C) 定義一個方法名稱爲 `doService()`

   (D) 定義一個方法名稱爲 `get()`

3. `HttpServlet` 是定義在哪個套件之中？

   (A) `javax.servlet`

   (B) `javax.servlet.http`

   (C) `java.http`

   (D) `javax.http`

4. 你在 web.xml 中定義了以下的內容：

```
<servlet>
    <servlet-name>Goodbye</servlet-name>
    <servlet-class>cc.openhome.LogutServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>GoodBye</servlet-name>
    <url-pattern>/goodbye</url-pattern>
</servlet-mapping>
```

哪個 URL 可以正確的要求 Servlet 進行請求處理？

(A) /GoodBye

(B) /goodbye.do

(C) /LoguotServlet

(D) /goodbye

5. 在 Web 容器中，以下哪兩個類別的實例分別代表 HTTP 請求與回應物件？

(A) `HttpRequest`

(B) `HttpServletRequest`

(C) `HttpServletResponse`

(D) `HttpPrintWriter`

6. 在 Web 應用程式中，何者負責將 HTTP 請求轉換爲 `HttpServletRequest` 物件？

(A) Servlet 物件

(B) HTTP 伺服器

(C) Web 容器

(D) JSP 網頁

7. 在 Web 應用程式的檔案與目錄結構中，web.xml 是直接放置在哪個目錄之中？

(A) WEB-INF 目錄

(B) conf 目錄

(C) lib 目錄

(D) classes 目錄

8. 你在 web.xml 中定義了以下的內容：

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <java-class>cc.openhome.HelloServlet</java-class>
</servlet>
<servlet-mapping>
    <mapping-name>HelloServlet</mapping-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

這個 web.xml 中的定義有哪些錯誤？

(A) `<url-pattern>` 標籤中的設定一定要用 `.do` 作結尾

(B) `<mapping-name>` 標籤應改爲 `<servlet-name>`，結尾標籤名稱也要修改

(C) `<java-name>` 標籤應改爲 `<servlet-class>`，結尾標籤名稱也要修改

(D) `<servlet>` 標籤應改爲 `<servlet-definition>`，結尾標籤名稱也要修改

9. 在 MVC 架構中，誰負責通知應用程式客戶端，應用程式本身有狀態改變？

(A) 模型（Model）

(B) 視圖（View）

(C) 控制器（Controller）

10. MVC 與 Model 2 架構最大的差別在於？

(A) Model 2 架構的視圖是由 HTML 組成

(B) Model 2 架構中的模型無法通知視圖狀態已更新

(C) MVC 架構是基於請求／回應模型

(D) MVC 架構只能用於單機應用程式

## 實作題

1. 撰寫一個 Servlet，當使用者請求該 Servlet 時，顯示使用者於幾點幾分從哪個 IP（Internet Protocol）位址連線至伺服器，以及所發出的查詢字串（Query String）。

   提示：請查詢一下 `ServletRequest` 或 `HttpServletRequest` 的 API 說明文件，了解有哪些方法可以使用。

2. 撰寫一個應用程式，可以讓使用者藉由在表單網頁上輸入名稱、密碼，若名稱爲 "caterpillar" 且密碼爲 "123456"，則顯示一個 HTML 頁面回應並有「登入成功」字樣，否則顯示「登入失敗」字樣，並有一個超鏈結連回表單網頁。注意！不可在網址列上出現使用者輸入的名稱、密碼。